Escuela
Politécnica
Superior

Predicción Bursátil con sistemas neurales

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Miguel Ángel García Gandía

Tutor/es:
Juan Ramón Rico Juan

Universitat d'Alacant
Universidad de Alicante

Noviembre 2020

Predicción bursátil con sistemas neurales

—

Author

Miguel Ángel García Gandía

Supervisors

Juan Ramón Rico Juan

Dpto. Lenguajes y Sistemas Informáticos

Trabajo final en grado de Ingeniería Informática

Alicante (España) - Noviembre 2020

Cuando mi limpiabotas compra en bolsa,
yo lo vendo todo.

John D. Rockefeller

Resumen

Este trabajo aborda el área de predicción de series temporales con Deep Learning. Más concretamente, se centra en la predicción del precio de la criptomoneda Bitcoin con redes neuronales.

Para hacer este estudio, analizaremos los datos de cotización del activo y, posteriormente, usaremos distintos modelos neuronales que compararemos entre sí.

Aunque en este documento planteamos el problema en el mercado de criptomonedas, este trabajo es extrapolable a los mercados clásicos bursátiles.

Agradecimientos

En esta sección quisiera agradecer a todos los que me han apoyado haciendo más llevadera mi etapa universitaria.

Comenzaré con un agradecimiento a mi tutor, Juan Ramón, por ayudarme y guiarme en los momentos más difíciles de este trabajo.

A mi familia, por su apoyo constante e incondicional, incluso en los momentos más difíciles, sin esperar nada a cambio. También mencionar a Amparo, Piera y Alfonso por mostrarme que el conocimiento es fascinante.

Para finalizar, a Jen, por esas charlas tan placenteras a la par que reparadoras. Sin ti habría abandonado la carrera hace tiempo. Mil gracias.

Índice general

Índice de figuras

Índice de cuadros

Listings

Capítulo 1

Introducción

Actualmente nos encontramos ante grandes avances tecnológicos relacionados con la Inteligencia Artificial. Este año, *OpenAI* decidió cancelar el lanzamiento de su software de *Natural Language Processing* (*NLP*) por considerar que habían logrado un nivel de generación de textos tan avanzados que podía usarse para la generación de noticias falsas.

Estos grandes avances se dan gracias al Deep Leaning, un área dentro del Machine Learning que a su vez se sitúa dentro del campo de la Inteligencia Artificial [1.1].

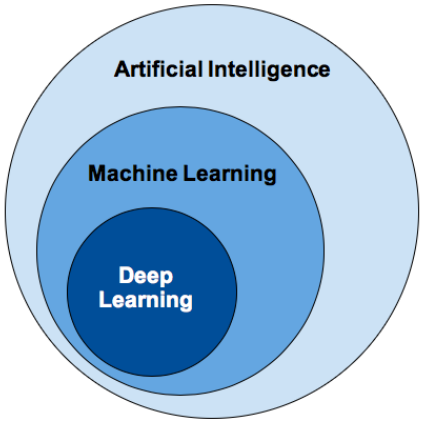

Figura 1.1: Ubicación del campo del Deep Leaning

Según las propiedades del sistema inteligente, podemos clasificarlo en dos grupos: la **inteligencia artificial débil** y la **fuerte**.

La inteligencia artificial débil es aquella que aborda un problema pequeño ya que es limitada. Casi todo el progreso dado en estos últimos años ha sido de éste tipo.

La inteligencia artificial fuerte es aquella que podría desarrollar cualquier tarea que

un humano pueda hacer, incluso hacerla mejor que un humano. Siendo críticos, creo que aún estamos bastante lejos de tener un sistema que pueda ser clasificado como inteligencia artificial fuerte.

En este trabajo nos centraremos en intentar desarrollar un sistema inteligente débil que se dedique a una tarea: predicción bursátil.

1.1. Mercados financieros y Machine Learning

La predicción de mercados financieros es uno de los principales retos de predicción de series temporales a los que nos enfrentamos en el campo del Machine Learning.

Podemos definir Machine Learning como la capacidad de un ordenador de aprender sin ser explícitamente programados.

Podemos clasificar éste campo en tres grandes grupos:

- **Aprendizaje supervisado**: Los datos que usamos para entrenar al modelo están etiquetados y contienen el comportamiento abstracto a imitar.
- **Aprendizaje no supervisado**: Los datos de entrenamiento no están etiquetados y el algoritmo intentará etiquetarlos.
- **Aprendizaje reforzado**: El modelo es un agente que explora un espacio desconocido y aprende a base de prueba y error. El agente debe resolver el problema de la mejor forma posible para recibir la mayor recompensa.

Cabe destacar que aunque nosotros nos centremos en el supervisado, existen multitud de trabajos donde usan el aprendizaje reforzado y no supervisado gracias a la propia naturaleza de los mercados financieros.

Los datos de estos mercados de cotización son enormes y no lineares ya que una estrategia que ha funcionado durante los últimos meses, puede dejar de funcionar porque el comportamiento del mercado ha cambiado. Es aquí donde entra en juego los modelos neuronales, dentro del área del Deep Learning, para intentar detectar esos cambios del mercado (en base al comportamiento del histórico) y actuar en consecuencia.

1.2. Deep Learning y redes neuronales

Como ya hemos mencionado anteriormente, el Deep Learning es un área dentro del Machine Learning. Un algortimo de este campo son las redes neuronales. Éstas las usaremos para intentar resolver el problema que abordamos.

Estas redes aprenden estructuras jerárquicas y abstraen patrones para comprender los datos de los que se alimentan. Esta comprensión de los datos se establecen en niveles de abstracción desde la entrada hasta la salida en formas cada vez más complejas.

Es por eso que a este campo se le denomina Deep Learning (Aprendizaje profundo), ya que no somos capaces de explicar estas abstracciones de una forma sencilla.

1.3. Motivación

Predecir el comportamiento del precio de los mercados financieros es el santo grial de la inversión en bolsa. Gracias al avance de la tecnología, podemos aplicar modelos de Deep Learning para buscar patrones ocultos en los datos y dar una mejor solución a este problema.

Actualmente existen equipos que se dedican a desarrollar algoritmos para aplicar al mercado bursátil y sacar rendimiento de este. Tan alta es la competencia que las empresas son capaces de invertir millones de dólares en hacer cables de fibra óptica que conecten su edificio directamente con el *backbone* que resuelve las operaciones de los mercados.

Todo esto, añadido al gran reto dentro del área de la predicción de series temporales que supone el comportamiento de estos mercados, es lo que ha hecho realidad el presente trabajo.

1.4. Objetivos

Los principales objetivos de este trabajos son:

- **Objetivo 0**: Conocer las diferentes tecnologías con las que se trabaja en este estudio.
- **Objetivo 1**: Conocer las diferentes arquitecturas con las que se trabajan.

- **Objetivo 2**: Comparar del rendimiento de los diferentes modelos aquí expuestos.
- **Objetivo 3**: Ventajas e inconvenientes a la hora de predecir los valores con una arquitectura u otra.
- **Objetivo 4**: Viabilidad de los modelos a la hora de sacar un rendimiento económico del mercado bursátil.

1.5. Estructura del trabajo

Para que la lectura se realice de un forma más cómoda, se listan a continuación los diferentes capítulos que tendrá el estudio y una breve descripción de cada uno de ellos.

- ***Capítulo 1: Introducción*** ⇒ Breve descripción y objetivos que tiene este trabajo.
- ***Capítulo 2: Breve introducción a los mercados financieros*** ⇒ Introducción a los mercados financieros y a los conceptos que abordaremos en los capítulos siguientes.
- ***Capítulo 3: Estado del arte*** ⇒ Últimos estudios publicados donde abarcan también el área de este trabajo.
- ***Capítulo 4: Tipos de modelos*** ⇒ Descripción de los modelos usados para la predicción bursátil.
- ***Capítulo 5: Análisis de datos*** ⇒ Descripción de los datos usados, análisis y normalización de éstos.
- ***Capítulo 6: Metodología*** ⇒ Descripción de los métodos usados para comparar los resultados obtenidos del siguiente capítulo.
- ***Capítulo 7: Experimentación*** ⇒ Datos obtenidos con la experimentación de cada modelo.
- ***Capítulo 8: Conclusiones y trabajos futuros*** ⇒ Conclusiones de los datos experimentados.

Capítulo 2

Breve introducción a los mercados financieros

Para poder entender la complejidad del problema a resolver tenemos que entender cómo funcionan los mercados financieros. Éstos no son objeto de estudio de este trabajo pero si que necesitamos unos conceptos básicos para poder entender algunos términos con los que se trabajará en el documento.

El análisis de los mercados financieros se dividen en dos grandes campos: el análisis fundamental y el análisis técnico. El precio de la moneda Bitcoin, respecto al dólar, estará marcado tanto por el análisis fundamental, como por el análisis técnico. Los negociantes (*traders*) son aquellas personas que compran y venden el activo del mercado haciendo que el precio vaya variando a lo largo del tiempo.

2.1. Análisis fundamental

El análisis fundamental se trata de predecir cuánto va a valer un producto a partir del estudio de las variables que afectan a su valor. Ejemplos de variables que afectan al valor de una acción o moneda pueden ser desde noticias financieras, resultados de ventas de una empresa, etc.

Este trabajo no se centra en el estudio del análisis fundamental pero si que cabe mencionar que hay muchos trabajos en predicción bursátil en este campo procesando cantidades enormes de datos. Un ejemplo utilizando este tipo de análisis sería una red neuronal que evalúa la confianza del mercado a partir de evaluar el sentimiento de los

tuits que contienen la palabra Bitcoin.

2.2. Análisis técnico

El análisis técnico se basa en el estudio de la acción del mercado siendo tres las acciones principales que se pueden estudiar:

- **Precio o cotización del mercado**: El precio que marca en un momento determinado el mercado. Existen muchos tipos de visualizaciones de precios pero esto lo dejamos para más adelante.
- **Volumen del mercado**: Cantidad o volumen de producto que se ha negociado en ese momento. Normalmente en las gráficas aparece como una barra debajo del precio de un momento determinado.
- **Interés abierto**: Solo disponible para productos financieros de futuros y opciones. En este trabajo al centrarnos con el par Bitcoin/Dólar no tendremos esta característica.

2.2.1. Teoría de Dow

La teoría de Dow es la que estableció las bases de lo que hoy conocemos como análisis técnico. Estas bases se pueden agrupar en estos cinco grandes puntos:

- El precio del mercado lo refleja todo.
- El mercado tiene tres tendencias: primaria, secundaria y menor.
- El volumen debe confirmar la tendencia
- Las tendencias principales tienen tres fases: fase de acumulación, fase de tendencia y fase de distribución.
- Se presume que una tendencia está en vigor hasta que da señales definitivas de que no marca nuevos mínimos o máximos dependiendo de la tendencia.

2.3. Velas japonesas

Las velas japonesas son una técnica de gráficos que se originó en Japón. Más tarde se implantó en el resto del mundo dada su facilidad de mostrar información relevante en cada tramo de tiempo de un mercado financiero.

Hay dos tipos de velas: velas alcistas y velas bajistas. La única diferencia entre ellas está en si el precio de apertura de la vela supera, o no, el precio de cierre de esa vela. Así se observa en la figura 2.1.

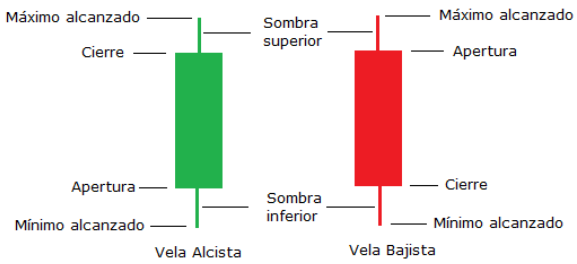

Figura 2.1: Características de las velas japonesas

Cada vela representa un rango de tiempo. Además, cada una de éstas puede descomponerse en otras velas que reflejan rangos menores de tiempo, pudiendo dar en algunos casos más información sobre qué pasará en la siguiente vela de temporalidad superior.

2.3.1. Retornos de vela

Denominamos al retorno como la variación del precio de un instante t y el instante t_n. Si el retorno es negativo, quiere decir que el precio entre esos dos instantes ha tenido un descenso. Si es positivo, indica que el precio ha subido.

En este trabajo hablaremos de retorno de la vela comparando siempre la vela actual con la anterior.

2.3.2. Conjuntos y formas de las velas japonesas

Aunque no es objeto de este trabajo estudiar todas las formas que existen de velas japonesas que pueden indicar cambios de tendencia u otro tipo de información, si que cabe mencionarlo puesto que esto nos es útil a la hora de saber analizar los datos con los que se trabajan.

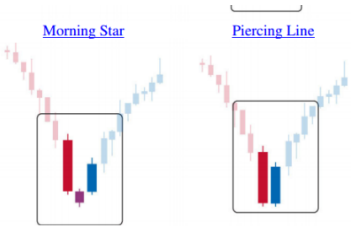

Figura 2.2: Ejemplo de conjuntos de velas que indican tendencias en el mercado

Como vemos en la figura 2.2 este conjunto de velas no suele pasar de las cinco velas y determina un cambio de tendencia a corto plazo.

2.3.2.1. GAP

Un ***GAP*** en bolsa ocurre cuando la cotización de un activo sufre una interrupción en un rango de precios donde no hay oferta ni demanda, provocando que la apertura de precio de la siguiente vela sea considerablemente distinta al precio en el que cerró la vela anterior.

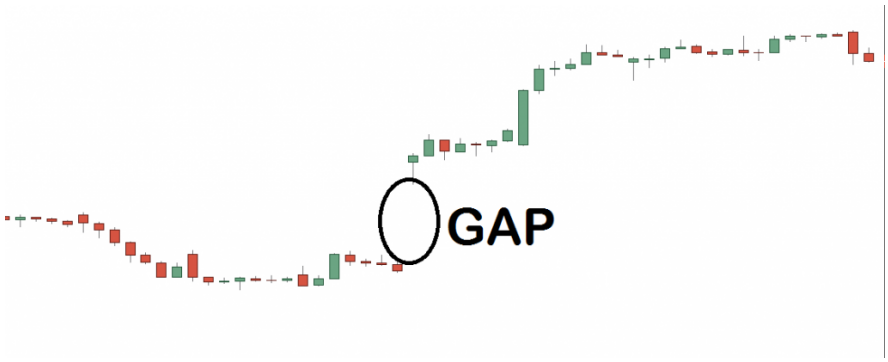

Figura 2.3: Ejemplo visual de *gap*

2.4. Conceptos básicos sobre tendencias

La tendencia es la dirección del mercado. Dependiendo de la dirección que tenga podremos clasificar al mercado como:

- **Mercado bajista**: El mercado crea nuevos mínimos en el precio.
- **Mercado alcista**: El mercado crea nuevos máximos en el precio.
- **Mercado indeciso**: La tendencia muestra lateralidad. No sea crean ni nuevos máximos ni nuevos mínimos en el precio.

Figura 2.4: Ejemplo de tendencias a lo largo del tiempo en el mercado Bitcoin/USDT

En la figura 2.4 podemos visualizar como en un tramo de tiempo una cotización puede establecer diferentes tendencias y además que esas tendencias se rompen normalmente con un gran volumen.

2.4.1. Figuras

El precio que cotiza un mercado puede hacer figuras a lo largo del tiempo más allá de las formas que hemos visto en el apartado anterior. Estas figuras son variadas y algunas de ellas están clasificadas como indicadores de cambio de tendencia o de continuación de ella como por ejemplo la figura 2.5.

Figura 2.5: Figura de cambio de tendencia: Doble suelo

Tampoco entraremos en detalle en esta cuestión pero es necesario saber, para su posterior análisis, que conforme evoluciona a lo largo del tiempo una cotización tenemos otra herramienta más de predicción.

2.4.2. Indicadores

Existen múltiples indicadores en la bolsa que vienen definidos por fórmulas matemáticas basadas en el precio y otras variables del mercado las cuales te pueden ayudar a definir mejor hacia dónde va la tendencia.

Estos indicadores se suelen agrupar en diferentes grupos dependiendo de lo que estén midiendo: volumen, volatilidad, tendencia, momento, etc.

Figura 2.6: Indicador RSI. En verde sobrecompra y en rojo sobreventa sobre un mercado

Uno de los indicadores más utilizados es el *Relative Strength Index* (**RSI**). Como podemos observar en la figura 2.6 el valor **RSI** marca grandes movimientos de sobrecompra o sobreventa cuando pasa de un cierto valor.

2.4.3. Volatilidad

En los mercados financieros, la volatilidad es la desviación estándar de la variación de un valor respecto al anterior en un horizonte temporal específico. Ésta mide la frecuencia e intensidad de los cambios del precio de un activo.

En los siguientes capítulos compararemos la volatilidad del mercado con la calidad de las predicciones que generan nuestros modelos.

2.5. Portafolio

El portafolio de inversión es una cesta de valores que se crea y configura de tal manera que te permita multiplicar tu ahorro, tu patrimonio financiero a lo largo de tu vida.

Más adelante veremos una forma de poder comparar el rendimiento de los modelos creados a través de hacer ***backtesting*** simulando un portafolio y viendo la efectividad de la red neuronal para acumular activos a traves de las acciones de compra-venta del mercado.

2.6. Drawdown

El drawdown mide el máximo retroceso en la curva de un portafolio de activos financieros. En el campo del trading, es una forma de medir el riesgo de una estrategia de mercado.

Figura 2.7: Drawdown en una gráfica de rentabilidad de un portafolio

2.7. Conclusiones del capítulo

Esta breve introducción es necesaria para poder entender con exactitud con qué datos y gráficas se van a trabajar en los siguientes capítulos.

El análisis técnico nos puede servir como herramienta a la hora de procesar los datos con los que alimentar a los modelos que usemos y a indicarnos que, dentro de la aleatoriedad que son los mercados financieros, existen ciertos indicadores que suelen indicar patrones del precio. El trabajo de nuestros modelos neuronales será el de detectar estos patrones para poder predecir qué pasará con el precio del mercado en velas posteriores.

Capítulo 3

Estado del Arte

Dentro de los algoritmos de Deep Learning que se usan para la predicción bursátil, en los últimos años hay diversos estudios que llegan a la conclusión de que las redes neuronales recurrentes son las que mejor realizan este problema ya que estamos abordando un problema de **análisis de series temporales**.

3.1. Series temporales

Una serie temporal es un orden cronológico de un conjunto de datos. La predicción de series temporales se encarga de buscar una función $\phi(x) = y$ donde x es $q_{t\text{-}n}$ instantes de tiempo (siendo n el nº de instantes de tiempo y t el instante actual) e y sea la predicción q_{t+1} tal y como se puede apreciar en la ilustración de abajo.

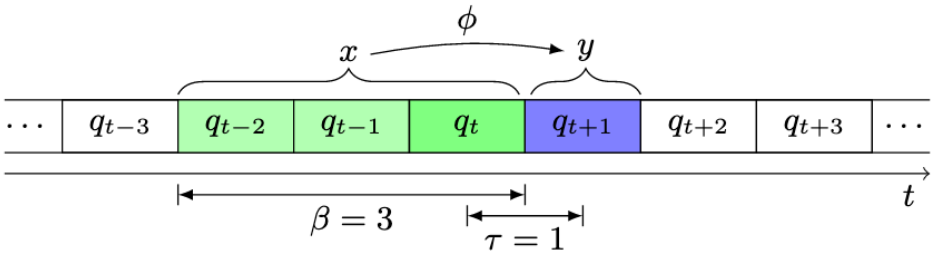

Figura 3.1: Ilustración de una serie temporal (Elend et al., 2020)

Podemos evaluar la aproximación a esta función ϕ comparando y con el valor q_{t+1} generado por nuestro modelo aplicando una función de error como por ejemplo *Mean Squared Error* (**MSE**).

3.2. Datos de entrada

Cada entrada a nuestra red neuronal será una dimensión más a tener en cuenta. Como vimos en la sección de Velas Japonesas 2.3, existen cinco dimensiones básicas para un periodo de tiempo en la cotización bursátil. De esta forma tendremos que para cada periodo q_{t-n} tendremos cinco dimensiones de entrada.

Estas entradas se pueden dar en dos tipos de valores: **valor absoluto** y **valor relativo**. En Liu et al. (2019) hace una comparación entre dos modelos iguales pero que cada uno usa un tipo de entrada diferente.

TABLE VIII
THE PREDICTION ACCURACY IN S&P500 INDEX.

Year	Year1	Year2	Year3	Year4	Year5	Year6	Average
Panel A.MAPE							
WSAEs-LSTM	0.012	0.014	0.010	0.008	0.011	0.010	0.011
C1D-LSTM	0.011	0.011	0.009	0.008	0.013	0.011	0.011
C1D-ROC	0.010	0.009	0.008	0.006	0.008	0.007	0.008
Panel B.Correlation coefficient							
WSAEs-LSTM	0.944	0.944	0.984	0.973	0.880	0.953	0.946
C1D-LSTM	0.962	0.973	0.988	0.986	0.860	0.958	0.955
C1D-ROC	0.965	0.979	0.988	0.982	0.949	0.976	0.973
Panel C.Theil U							
WSAEs-LSTM	0.009	0.010	0.006	0.005	0.008	0.006	0.007
C1D-LSTM	0.007	0.007	0.006	0.005	0.008	0.007	0.007
C1D-ROC	0.007	0.006	0.005	0.004	0.005	0.005	0.005

Figura 3.2: Resultados tabla por tipo de entrada (Liu et al., 2019)

En la figura 3.2 observamos los resultados de los dos modelos con diferentes valores. En este caso ***C1D-LSTM*** tiene entradas en valores absolutos mientras que ***C1D-ROC*** tiene entradas en valores relativos. En conclsuión, ***C1D-ROC*** tiene una leve mejora respecto a ***C1D-LSTM***.

En este y otros trabajos se introducen a la red otras características como indicadores técnicos (mencionados en el apartado 2.4.2) o datos macroeconómicos. En este trabajo nos centraremos solo en los datos de las velas japonesas.

3.3. Tipos de neuronas usadas en la actualidad

En Healton et al. (2018) se menciona como las *Long Short Term Memory* (**LSTM**) son capaces de dar una mejor solución debido a la incorporación de celdas de memoria. Aplicando a nuestro modelo varias capas de *LSTM* podemos predecir mejor los

comportamientos dinámicos que con otro tipo de neuronas como por ejemplo las redes neuronales recurrentes tradicionales.

Como hemos visto en los capítulos anteriores, una de las grandes dificultades que tienen las redes neuronales a la hora de predecir los mercados es el ruido. Llamamos ruido a aquellas entradas que son iguales pero dan resultados diferentes.

En Liu et al. (2019) se usa *1-D ResNet block* para intentar solucionar este tipo de escenarios que son naturales a cualquier cotización bursátil. También se añaden más dimensiones de entrada al modelo como por ejemplo datos macroeconómicos.

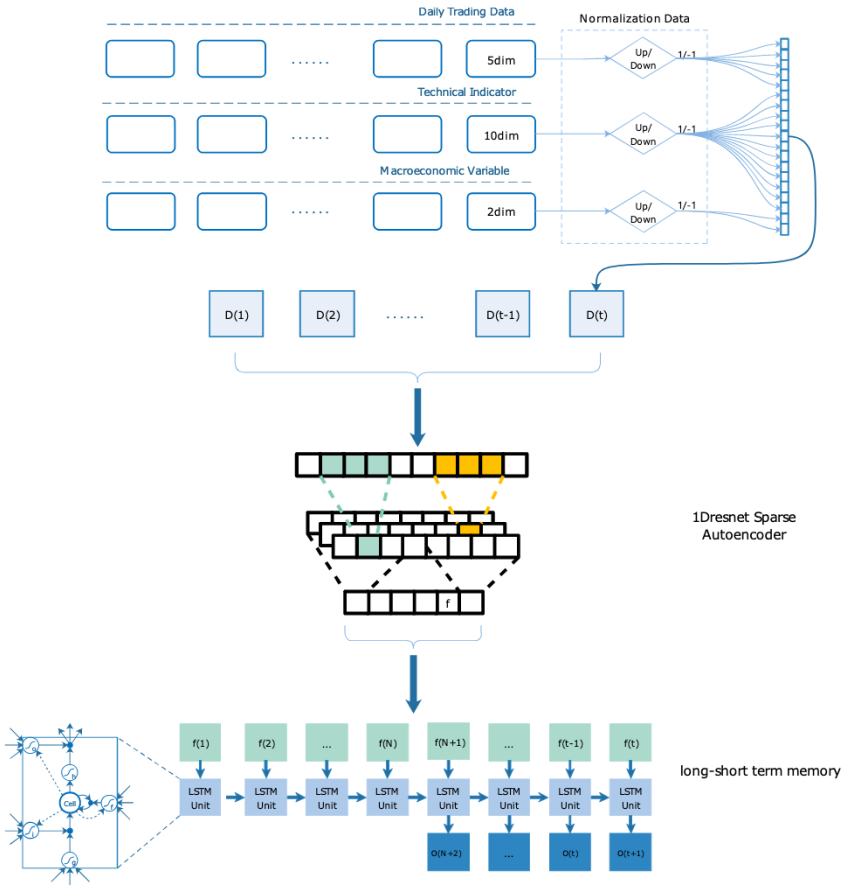

Figura 3.3: Proceso de extracción y predicción de características (Liu et al., 2019)

Este bloque *1-D ResNet* está formado por un *Sparse Autoencoder*. Dentro de los tipos de *autoencoder* que existen, las ventajas que presentan éstos son las de prevenir el *overfitting* y descubrir características ocultas en los datos de entrada mediante la activación de unas pocas neuronas por capa manteniendo el rendimiento de la red. De esta forma, podemos garantizar que el autoencoder realmente aprende representaciones latentes y no información redundante.

3.4. Funciones de error

Una de las medidas más importantes a la hora de entrenar modelos es el cálculo del error que posteriormente se retropropagará para cambiar los pesos de las neuronas. Este cálculo se hace a través de la función de error. En la inmensa mayoría de documentos acerca de predicción bursátil con redes neuronales se emplea la función de error MSE (Mean Squared Error).

Como curiosidad, en Doshi et al. (2020) crean su propia función de error para maximizar el funcionamiento correcto del modelo. La función de error es:

$$\mathcal{L}(x^*_{t+1}, x_{x+1}) = \|x^*_{t+1} - x_{t+1}\|^2 \quad (x_{t+1} - x_t) \cdot (x^*_{t+1} - x_t) < 0$$

Donde x^* es la predicción del mercado y x es el valor del mercado real. De esta forma penalizamos más cuando la red neuronal haga una predicción donde la tendencia de esa predicción sea contraria a la tendencia que realmente ocurre posteriormente.

3.5. Evaluación y rendimiento

Por último, otra tarea a realizar es la forma de comparar un modelo respecto a otro o evaluar su rendimiento.

3.5.1. Medidas Estadísticas

Una forma de comparar un modelo con respecto a otro es utilizar diferentes mediciones estadísticas. En Liu et al. (2019) se usa *Mean absolute percentage error* (**MAPE**), *Theil U* y una correlación lineal entre ambas mediciones. Esto se puede observar en la figura 3.2.

3.5.2. Comparación entre modelos

Otra forma que existe para comparar modelos, es usar una función de error, por ejemplo MSE, y comparar los resultados obtenidos con los de otro modelo.

El modelo base que se usa en Doshi et al. (2020) es **ARIMA**, *Auto-regressive*

integrated moving average, y se compara con un modelo LSTM.

Company	Mean Squared Error ARIMA	LSTM model
Ford	0.025	0.1185
Tesla	196.056	4624.22
Toyota	1.877	4.87
General Motors	0.34	0.413

Figura 3.4: Comparación de MSE entre modelo ARIMA y LSTM (Doshi et al., 2020)

En este caso en concreto, no son representativos los resultados obtenidos ya que con LSTM, aún dando mayor error, es capaz de producir un mejor rendimiento que el modelo ARIMA ya que al ser un modelo no lineal se adapta mejor a los datos de entrada. Esto lo podemos observar en la figura 3.5.

3.5.3. Backtesting

Una última forma de comparar el rendimiento de los modelos entrenados es simular el comportamiento mediante ***backtesting***. Así podemos aplicar acciones de compra-venta dependiendo de la predicción del modelo a evaluar y luego comparar con lo que hubiese pasado realmente.

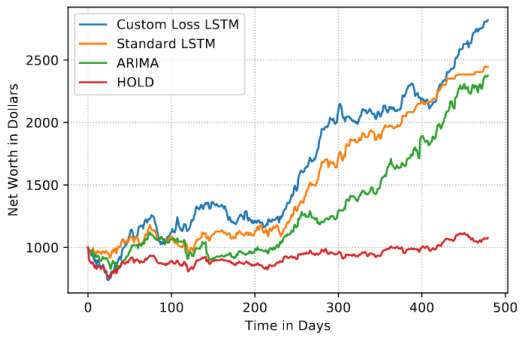

Figura 3.5: Portafolio comparando diferentes estrategias. (Doshi et al., 2020)

Como se observa en la figura 3.5, se compara la evolución de una cartera en bolsa aplicando la función de error vista en el apartado 3.4. Además se compara con el modelo **ARIMA** visto anteriormente (apartado 3.5.2).

Capítulo 4

Tipos de modelos

Un modelo no es más que un esquema, generalmente matemático, que intenta reproducir el comportamiento de un sistema. Podemos dividir estos modelos en dos grandes grupos: **modelos lineales** y **no lineales**.

4.1. Modelos Lineales

Son modelos matemáticos que relacionan diferentes variables que tienen una dependencia lineal entre ellas.

4.1.1. Regresión lineal

Este modelo matemático es capaz de aproximar la relación de entre variables a través de un plano o hiperplano.

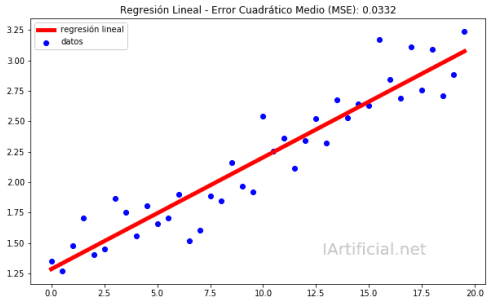

Figura 4.1: Ejemplo de regresión lineal

En la figura 4.1 podemos observar como se relacionan x e y a través de una recta gracias a la distribución de éste conjunto de datos.

4.2. Modelos no lineales

Son aquellos que relacionan diferentes dimensiones aplicando una función polinómica mayor que grado uno. Pueden hacer una mejor relación de las dimensiones de entrada pero normalmente a un costo computacional mucho mayor.

En este trabajo nosotros nos centraremos con los modelos neurales los cuales son no lineales.

4.2.1. Redes neuronales

Una red neuronal está compuesta por conjunto de neuronas o perceptrones. Este conjunto a su vez está formado por grupos de neuronas llamadas capas.

4.2.1.1. Perceptrón

El perceptrón es la unidad básica de una red neuronal. Está formado por una función lineal y una función de activación como podemos ver en la figura 4.2.

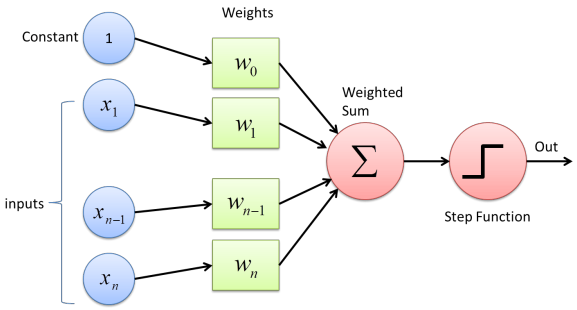

Figura 4.2: Perceptrón

A nivel matemático:

$$1 : y = W^T X + b \qquad 2 : f(x) = \begin{cases} 0 for x < 0 \\ x for x \geq 0 \end{cases}$$

Siendo la formula uno la función lineal la cual viene expresada por vectores y la formula dos una función de activación, siendo en este caso la función *Rectified Lineal Unit* (**ReLU**).

Cabe destacar que el bias (b), en algunas formulas, lo tratan como el elemento W_0 de los pesos del perceptrón.

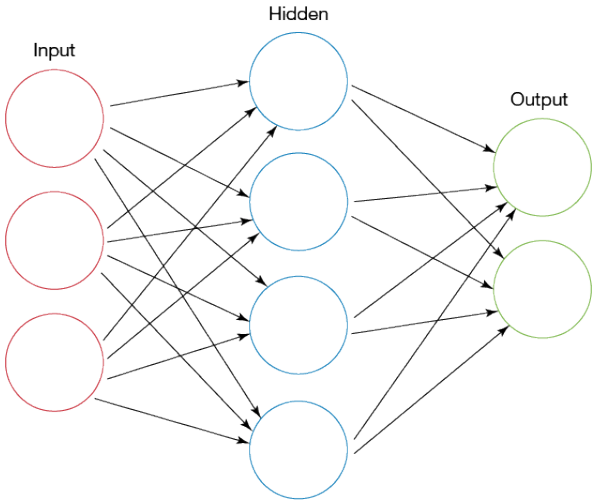

Figura 4.3: Capas de una red neuronal

Estos perceptrones se ubican por capas conectando cada perceptrón con todos los perceptrones de la capa siguiente dando lugar a la red neuronal. Como se observa en la figura 4.3 clasificamos las capas de una red neuronal en tres grandes grupos:

- **Capa de entrada**: Es la capa que recibe los datos de entrada de nuestra red.
- **Capa oculta**: Se denominan al conjunto de capas que están entre la capa de entrada y la capa de salida.
- **Capa de salida**: Es la capa de salida de nuestra red.

4.2.1.2. Funcionamiento de una red neuronal

Como pasa con el perceptrón, necesitaremos optimizar los pesos de nuestra red neuronal para que pueda generar un plano n-dimensional con el cual pueda resolver el problema que le hemos planteado (figura 4.4). Una red neuronal, al ser un conjunto de capas, el error generado a la salida no solo depende de una sino de varias. Esta naturaleza hace que sea bastante más costoso el cálculo de la responsabilidad de cada capa en el resultado final.

Para ello se utiliza un algoritmo llamado ***backpropagation***. Éste modifica los pesos de cada neurona dependiendo de cómo varía el error de la función de coste de la salida de la red neuronal.

Así pues, lo que tenemos que calcular es cómo varía el coste ante cada parámetro dentro de la red neuronal. Para eso, el algoritmo de backpropagation, hace uso de cuatro expresiones las cuales aplica desde la última capa hasta la primera. La primera

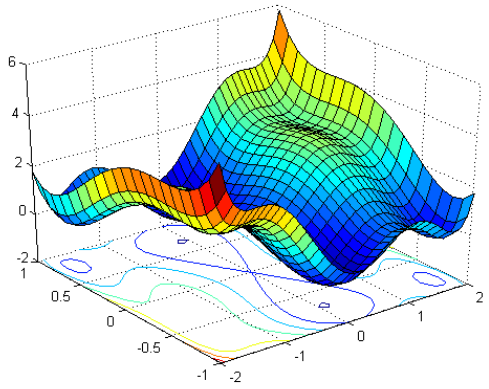

Figura 4.4: Hiperplano neuronal

expresión es calcular el error de la ultima capa ∂^L.

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$$

Esta expresión se usará solo en el cálculo de la última capa donde el error ya pertenece a la función de coste. La segunda es retropropagar el error a la capa anterior denominada ∂^{L-1}.

$$\delta^{L-1} = W^l \delta^L \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}}$$

Esta fórmula se usa en el resto de capas de nuestra red que el error dependerá de la capa anterior. La tercera y cuarta formula sería calcular las derivadas de la capa anterior usando el error respecto al bias y a los pesos.

$$\frac{\partial C}{\partial b^{L-1}} = \delta^{l-1} \qquad \frac{\partial C}{\partial w^{L-1}} = \delta^{l-1} a^{L-2}$$

Estas ecuaciones nos permiten calcular el error para el parámetro de bias y de los pesos de esa capa.

4.2.2. Arquitecturas neuronales

En lo que se refiere a las arquitecturas de red neuronal, es decir, la forma en la que están organizadas las neuronas en la red, podemos hacer un listado de las principales clases de redes que existen. Cabe destacar que existen multitud de clases pero aquí solo se explicarán aquellas que vamos a usar para la predicción bursátil.

4.2.2.1. Multilayer Perceptron

La clase MLP, *multilayer perceptron*, consiste en una red neuronal de mínimo tres capas (*input layer, hidden layer, output layer*), donde cada una de sus capas, excepto la de *input layer*, están compuestas de neuronas.

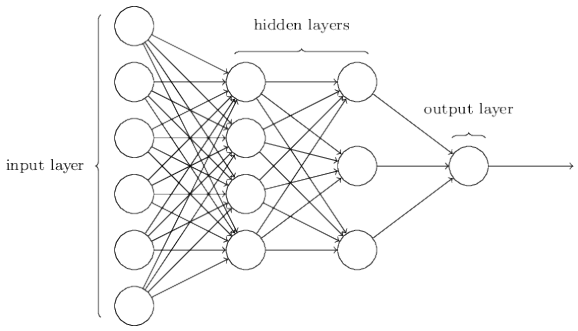

Figura 4.5: Esquema de arquitectura MLP

Todas las capas están completamente conectadas donde cada perceptrón está conectado con todos los perceptrones de la capa anterior y de su capa posterior. Además cada capa puede tener una función de activación diferente.

Al aplicar capas de perceptrones, podemos resolver problemas linealmente no separables que con el perceptrón no es posible realizar correctamente.

4.2.2.2. Redes Convolucionales

Las redes convolucionales (**CNN**) adquieren un uso muy extendido en el reconocimiento de imágenes dado el potencial que tienen para extraer características de éstas. El funcionamiento de estas redes se basa en tomar como entrada una imagen y reconocer patrones desde los más sencillos a los más complejos para dar una solución. Esta imagen se representa como una matriz de números que viene definida por la altura, el ancho y la dimensión. Por ejemplo, una imagen de 10x10x1 (Ancho x Alto x Dimensión) no es más que una imagen de 10 x 10 píxeles en escala de grises. Si la dimensión fuese 3, tendríamos la imagen en color (**RGB**).

Técnicamente, las CNN pasan una serie de filtros convolucionales y de reducción como se aprecia en la figura 4.6.

La convolución es el primer paso para extraer características de una imagen. Esta técnica preserva la relación entre los píxeles usando pequeños cuadrados que recorren toda la imagen y que se multiplican por un filtro (figura 4.7), dando como resultado

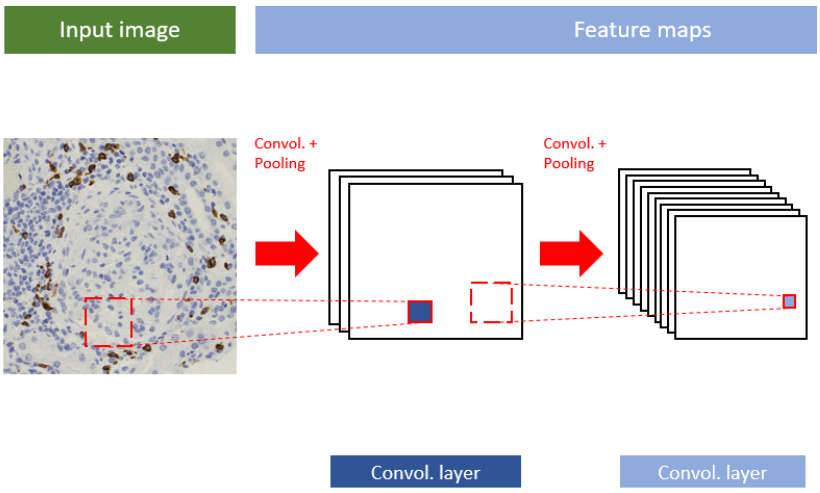

Figura 4.6: Esquema de una CNN

una característica convolucional.

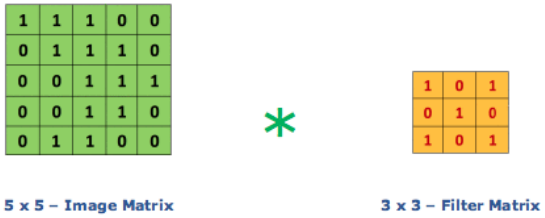

Figura 4.7: Imagen y Filtro a aplicar en la convolución

Estos filtros, también llamados ***kernels***, se usan en segmentación de imágenes y hay diversos tipos (detección de ojos, filtro gausiano, ...). Cada filtro será una matriz diferente a aplicar.

La ventana amarilla de la figura 4.8 es una iteración de ese proceso convolucional donde escoge la ventana de píxeles marcados y los multiplica por la matriz del filtro (subíndices de las casillas marcados en rojo) dando le resultado final.

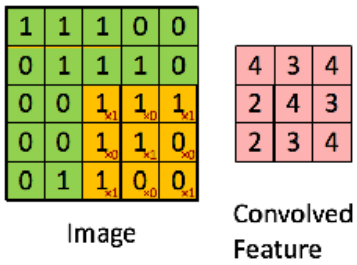

Figura 4.8: Proceso de convolución

La ventana se desplaza el número de píxeles que nosotros queramos. A este número de píxeles se le denomina ***stride***. Por ejemplo, si nosotros seleccionásemos un stride de 2, tendríamos una convolución como la de la figura 4.9.

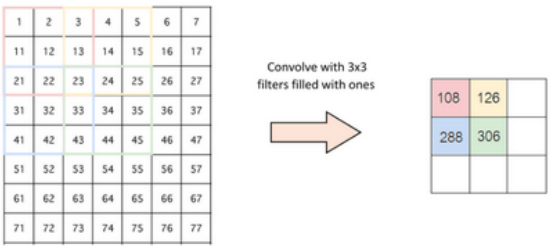

Figura 4.9: Proceso de convolución con stride 2

Habrá veces que dependiendo de las dimensiones de la imagen y del stride seleccionado puede que haya ventanas de la convolución que no se adapten al tamaño de la imagen.

Tenemos dos formas de solucionar este problema. Por un lado podemos rellenar de ceros aquellas partes de la ventana convolucional que no tienen valor. A esta operación se le denomina padding (figura 4.10). Por otro lado, podemos no aplicar la convolución a aquellas ventanas las cuales tengan alguno de sus píxeles fuera de la dimensión de la imagen.

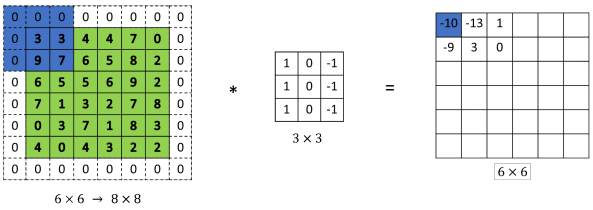

Figura 4.10: Proceso de convolución con padding

Una vez hemos aplicado la convolución, podemos reducir el tamaño del resultado mediante la aplicación de operación de *pooling*. Como hemos visto anteriormente con las convoluciones, en el pooling también tienes que declarar el tamaño del pooling y un stride (figura 4.11).

Los tres tipos de pooling más usados son:

- **Max Pooling**: Da como solución el valor más alto de la matriz.
- **Average Pooling**: El resultado es la media de todos los valores de la matriz.
- **Sum Pooling**: Suma todos los resultados de la matriz.

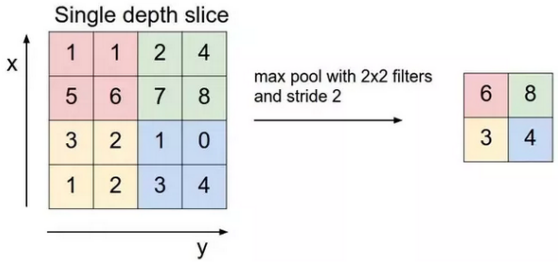

Figura 4.11: Proceso de pooling con Max Pooling

En las redes neuronales convolucionales, este proceso de convolución y pooling se apila en capas. Cuantas más capas convolcionales tenga, más caracerísticas complejas podrá abstraer la red.

Al final de toda red de este tipo, existe un aplanamiento de esta matriz (*flatten*), para pasar esas matrices convolucionales a vectores que se puedan conectar a un *MLP* que termine haciendo el trabajo de clasificación o regresión, gracias a la ayuda de la *CNN* extrayendo las características más importantes.

Hemos visto como en el Estado del Arte (apartado 3) se usan este tipo de redes neuronales en el campo de la predicción bursátil ya que ayudan a quitar el ruido existente en los datos.

4.2.2.3. TCN

Las redes convolucionales temporales (**TCN**) son una variación de las redes neuronales convolucionales para tareas de modelado de secuencias.

Esta arquitectura tiene dos puntos clave que la hacen atractiva para aplicarlas a éste campo. Por un lado, está diseñada para que la salida sea la entrada en el siguiente paso de tiempo. Por otro lado, el elemento de salida t solo se convoluciona con los elementos $t-n$ anteriores lo que puede resultar interesante para competir con las redes recurrentes que veremos en el próximo apartado.

4.2.2.4. LSTM

Las LSTM son un tipo especial de redes recurrentes. La característica principal de las redes recurrentes es que la información que se recibe puede ser almacenada para usarla en otra predicción. Esta característica las hace muy adecuadas para manejar series temporales. Mientras que las primeras redes recurrentes que se crearon no podían

mantener esa memoria a largo plazo, las LSTM si pueden, por lo que se podría decir que las hace especialmente favorables a la predicción bursátil.

Cómo bien dicen sus creadores Hochreiter and Schmidhuber, las LSTM están explícitamente diseñadas para resolver el problema de dependencia a largo plazo. Las redes recurrentes, incluídas las LSTM, se estructuran de forma concatenada para pasarse información tal y como se muestra en la siguiente figura 4.12.

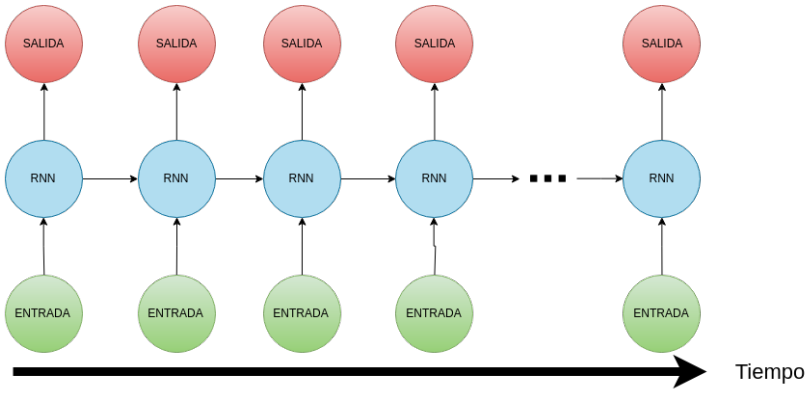

Figura 4.12: Esquema de una RNN

Estas redes están compuestas por una red neuronal interna que recoge la información de la neurona anterior y de la entrada actual y la salida de esta red se la pasa como salida de la neurona y a su vez pasa esa información a la neurona de la derecha.

Lo que diferencian las LSTM del resto, es en vez de tener solamente una red neuronal tiene cuatro como se puede ver en la figura 4.13.

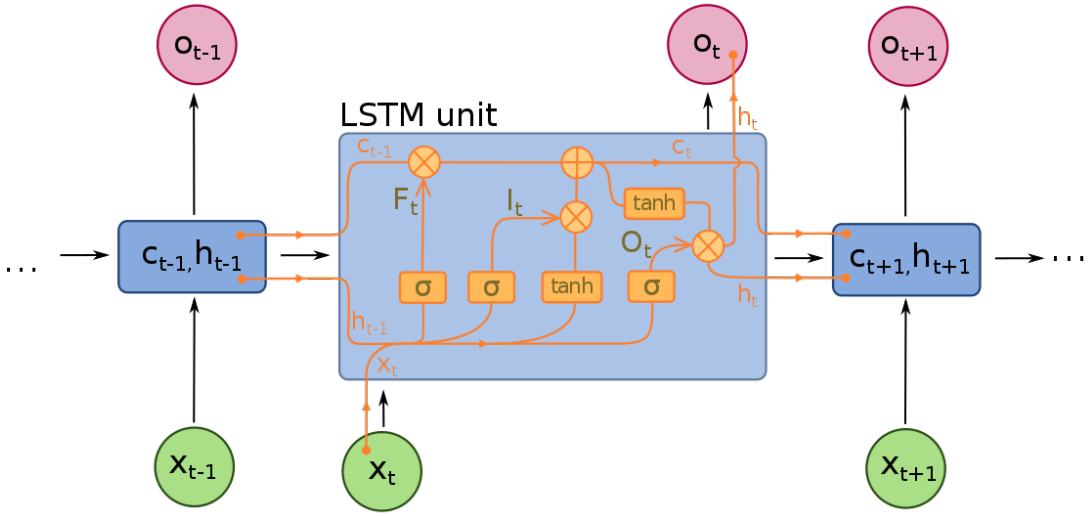

Figura 4.13: Esquema de una LSTM

Primero hay que decidir qué es lo que se va a borrar o no de la celda de estado.

Para ello mira h_{t-1} y x_t y lo pasa por una capa sigmoidea dando F_t. Un 1 en esta salida significa guardarse completamente C_{t-1} mientras que un 0 significa borrarlo completamente. Este F_t modificará el estado C_{t-1} que proviene de la LSTM anterior. La formula para el cálculo de F_t es:

$$F_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

El siguiente paso es ver qué es lo que vamos a guardar en la celda de estado. Esto se realiza en dos apartados. El primer apartado será decidir qué valores se actualizarán a partir de la capa sigmoidea de puerta de entrada (i_t). El segundo apartado será crear un vector de candidatos que podrían añadirse al estado (c_t). Una vez se actualice el estado C_{t-1} a partir de los valores C_{t-1}, i_t y c_t obtendremos el estado de nuestra celda actual C_t.

$$i_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$C_t = F_t * C_{t-1} + i_t * c_t$$

Este estado C_t servirá como estado C_{t-1} a la LSTM siguiente y para pasarlo por una capa tangente hiperbólica que vuelva el rango de valores desde -1 a 1. Este resultado lo multiplicaremos por el resultado de una capa sigmoidea O_t la cual decidirá que partes del estado de la celda vamos a generar. El resultado será h_t el cual será la salida de esa LSTM y a su vez será la entrada h_{t-1} de la siguiente LSTM.

$$O_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$h_t = O_t * tanh(C_t)$$

4.2.2.5. Gated Recurrent Unit

Como hemos visto, las LSTM son bastante efectivas para recordar a largo plazo. Otro tipo de neurona creada para solucionar estos problemas de las redes recurrentes tradicionales, son las Gated Recurrent Unit (GRU).

Presentadas en Cho et al. (2014), las GRU pueden ser consideradas como una

variación de las LSTM. En muchos problemas tanto las GRU como las LSTM dan resultados prácticamente iguales.

En este caso las GRU usan dos puertas en vez de cuatro como las LSTM. Estás puertas se usan para guardar información de hace mucho tiempo o para borrar información innecesaria al igual que pasa con las LSTM.

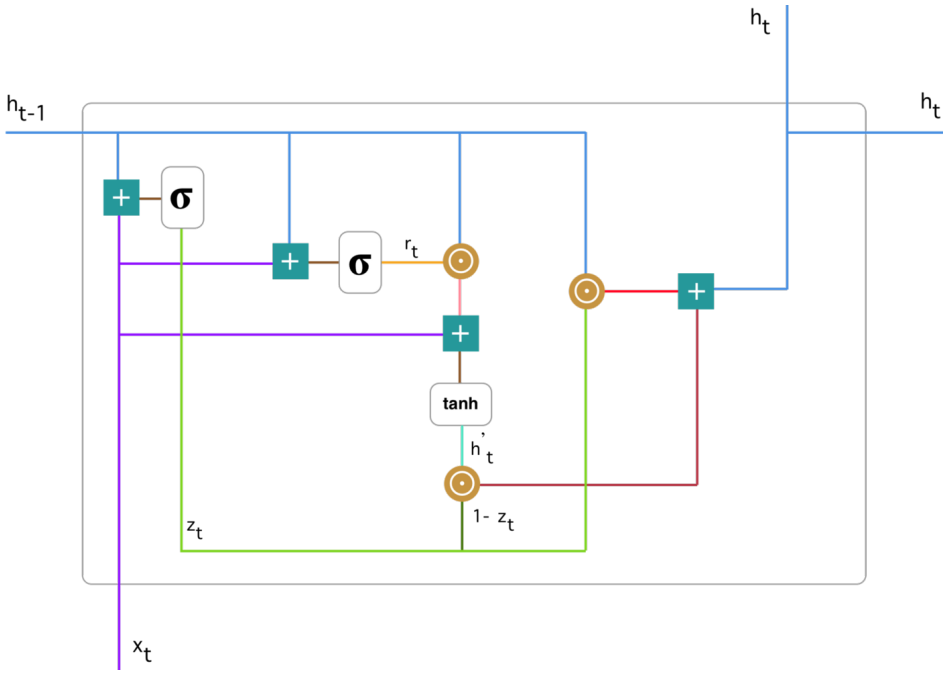

Figura 4.14: Esquema de una GRU

Como podemos ver en la figura 4.14, el primer paso será calcular z_t. Para ello tenemos que sumar x_t y h_{h-1} y pasarlo por una capa sigmoidea.

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

Siendo $W^{(z)}$ y $U^{(z)}$ los propios pesos de x_t y h_{t-1} respectivamente. Con esta operación el modelo puede decidir qué información decide quedarse del pasado al igual que pasa en las LSTM.

El siguiente paso es olvidar aquella información del pasado que no queramos. Esta puerta existe también en las LSTM. En el modelo GRU la denominaremos como r_t y se calcula de la misma forma que z_t.

$$r_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

Una vez tenemos calculados r_t hay que calcular h'_t. Para ello tendremos que multi-

plicar las matrices r_t y h_{t-1} y sumar el resultado con x_t. Posteriormente lo pasaremos por una capa tangente hiperbólica para aplanar los valores entre -1 y 1.

$$h_t^{'} = tanh(Wx_t + r_t \odot Uh_{t-1})$$

Esta $h_t^{'}$ que hemos calculado servirá para ver qué información del pasado es relevante y hay que almacenarla en esta celda.

El último paso es calcular h_t. Este valor será la salida de la GRU y además será usada como h_{t-1} por la GRU posterior. La idea de h_t es la de que la salida se vea afectada por aquello que es de interés de la celda de memoria de la GRU una vez se ha actualizado que es de interés de la salida de la GRU anterior. Se calcula de esta forma:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t^{'}$$

Capítulo 5

Análisis de datos

El conjunto de datos usado serán del par BTCUSDT en la temporalidad de cinco minutos. Esta temporalidad se ha escogido para tener un alto detalle del movimiento del precio y, así, poder sacar el mayor rendimiento posible en el menor plazo de tiempo.

Figura 5.1: Movimientos en contra de la tendencia principal en cinco minutos

5.1. Estructura de los datos

Las características con las que trabajamos en el *dataset* son seis como vemos en el cuadro 5.1.

En la figura 5.2 se observa como existe una correlación entre el volumen y el movimiento del precio.

Timestamp	Open	High	Low	Close	Volume
1517978700	7438.00	7499.00	7415.16	7465.00	197.998663
1517979000	7470.88	7550.00	7433.61	7522.00	251.023185
1517979300	7522.10	7531.63	7450.00	7460.00	153.566643
1517979600	7451.01	7480.00	7434.28	7456.20	119.214825
1517979900	7466.99	7474.92	7375.02	7423.67	168.354553

Cuadro 5.1: Pequeña muestra de los datos BTCUSDT en 5 minutos.

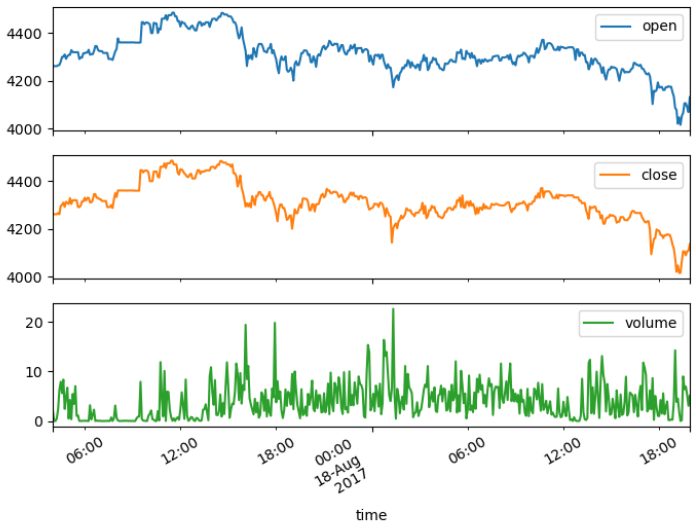

Figura 5.2: Pequeña muestra gráfica de los datos BTCUSDT en 5 minutos *(open, close y volume)*

5.2. Procesado de los datos

Al intentar realizar modelos de predicción con el conjunto de datos de forma absoluta, aunque los datos estén escalados, pequeñas variaciones de decimales en la predicción hace que entre las velas de entrada y las velas predichas se forme un GAP (apartado 2.3.2.1). Este hecho hace que los valores absolutos no sean viables para intentar sacar rendimiento del mercado.

La solución a este problema ha sido transformar los datos absolutos en retornos (apartado 2.3.1).

Las ventajas de usar los retornos de la vela, en vez de los valores intrínsecos, son que nos centramos en las variaciones del precio y, aunque se pierde ese valor absoluto, no es tan importante para sacar rendimiento en el mercado en temporalidades bajas como es nuestro caso.

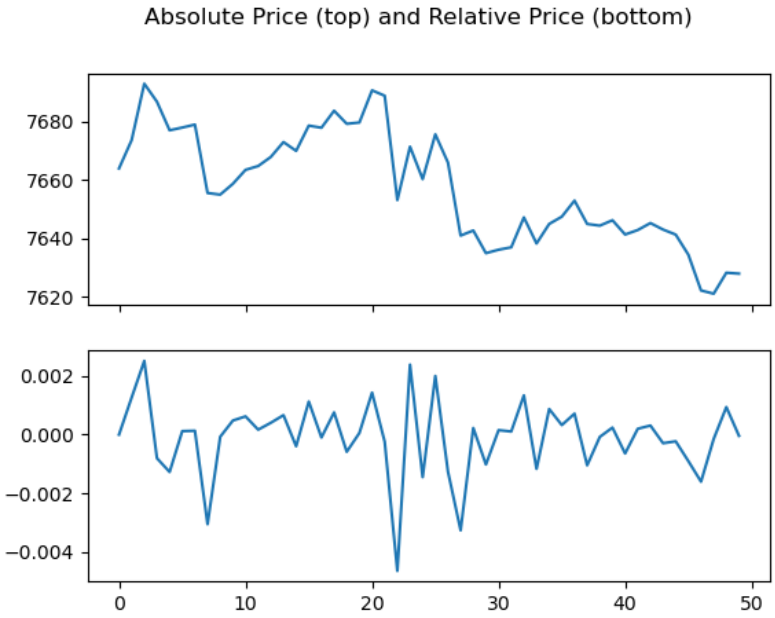

Figura 5.3: Ejemplo gráfico de serie temporal expresado de forma absoluta y relativa

Finalmente, se quitan posibles defectos en el *dataset* (valores nulos o infinitos) sustituyendo esos valores por una interpolación lineal de los valores más cercanos (figura 5.7).

5.2.1. Distribución de los datos

Una vez aplicada las transformaciones del conjunto de datos original, vamos a observar como se distribuyen las características de las velas (*open*, *high*, *low*, *close* y *volume*).

En la capítulo de *Experimentación*, usaremos algoritmos de normalización aplicados a los datos de este apartado. El objetivo es evaluar si con algoritmos de escalado que no les afecte los valores extremos de la distribución es posible obtener mejores resultados.

5.2.2. Timestamp

Para reducir el ruido del mercado utilizamos la característica temporal de cada vela. Asignamos cada valor de minuto y hora a un valor del rango seno y coseno tal y como se aprecia en la figura 5.6. De esta forma pasamos un valor de *timestamp* a rangos normalizados e introducimos cierta relación entre minutos y horas. Ésta relación que puede existir entre el comportamiento de un mercado en horarios similares es lo que se

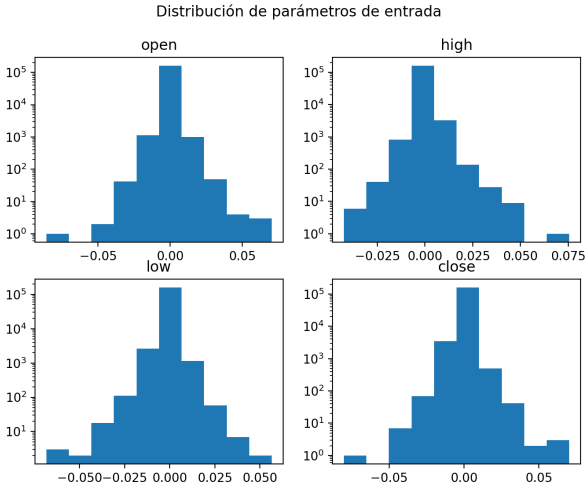

Figura 5.4: Distribución normal del retorno de las velas

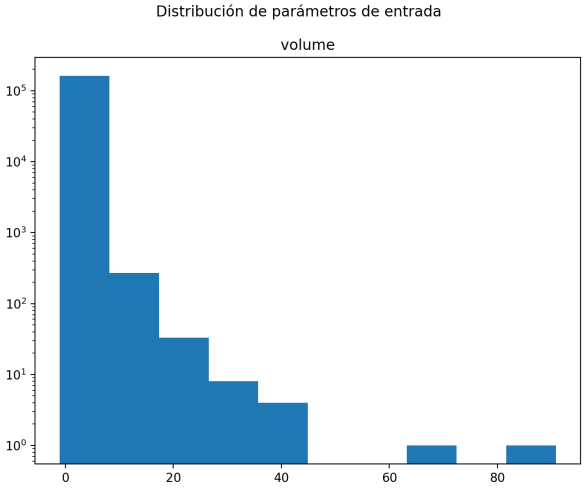

Figura 5.5: Distribución geométrica del volumen

denomina estacionalidad.

5.2.3. Dataset procesado

Las características de cada vela definidas como entrada a los modelos finalmente serán nueve (cuadro 5.2).

5.3. Proceso de evaluación de modelos

Al ser el *dataset* una serie temporal, con un comportamiento dinámico, hace que la forma habitual de generar los datos de entrenamiento, validación y test no sea la

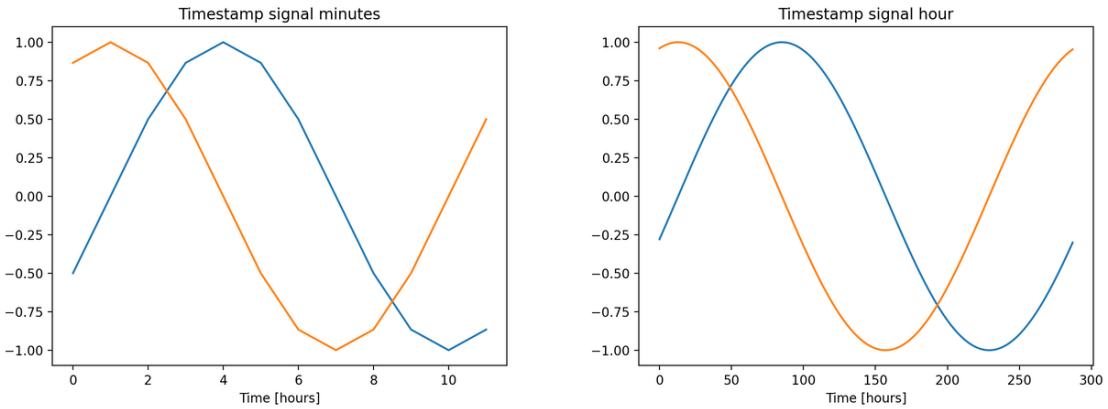

Figura 5.6: Transformación de Timestamp a Seno y Coseno

feature	count	min	max
Open	112748.0	-0.085924	0.070345
High	112748.0	-0.042523	0.075630
Low	112748.0	-0.068392	0.049440
Close	112748.0	-0.080102	0.070346
Volume	112748.0	-1.000000	90.798003
Minute Sin	112748.0	-1.000000	1.000000
Minute Cos	112748.0	-1.000000	1.000000
Hour Sin	112748.0	-1.000000	1.000000
Hour Cos	112748.0	-1.000000	1.000000

Cuadro 5.2: Valores de velas de forma relativa

adecuada para evaluar objetivamente nuestros modelos.

Para resolver este problema, se ha aplicado la técnica *Blocked Cross-Validation*. Ésta técnica se basa en crear partes más pequeñas a partir del conjunto de datos inicial para poder evaluar los modelos en distintos comportamientos del mercado.

En nuestro caso, crearemos diez partes de igual longitud tal y como se puede observar en la figura 5.7. A cada situación o parte se le denominará **iteración**.

5.3.1. Coeficiente de relación Spearman

Para asegurarnos que el comportamiento de los diferentes bloques en los que hemos dividido el *dataset* son diferentes, es decir, no están relacionados, vamos a aplicar una medida denomindada *Coeficiente de correlación de Spearman* (ρ).

Hemos escogido este coeficiente de relación respecto al de *Pearson*, porque *Spearman* es más robusto a la hora de soportar ciertos desvíos. Ésto nos vendrá ideal para ciertos

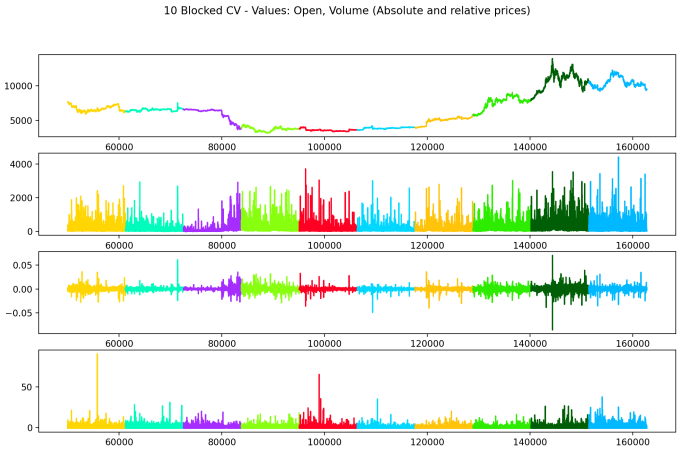

Figura 5.7: Valores open y volume, en absoluto y relativo, divididos en diez iteraciones

valores del volumen (cuadro 5.2) que cumplen estas características.

El coeficiente de relación de *Spearman*, ρ, viene dado por la expresión:

$$\rho = 1 - \frac{6\Sigma D^2}{N(N^2 - 1)} \tag{5.1}$$

Donde N es el número de parejas de datos y D es la diferencia entre los correspondientes estadísticos de orden de $x - y$.

La interpretación de *Spearman* es la misma que la de *Pearson*: el valor 0 indica que la correlación es mínima mientras que los valores cuanto más tiendan a 1 o -1 quiere decir que la correlación es máxima.

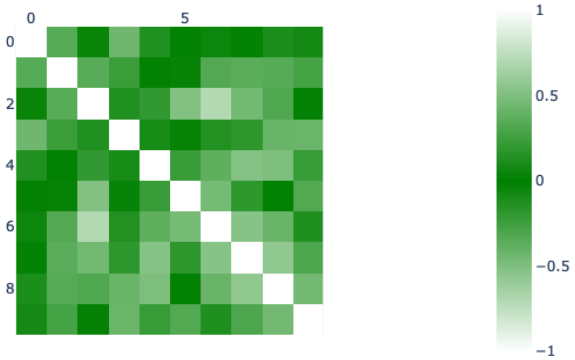

Figura 5.8: Correlación de Spearman entre los distintos bloques

Se entiende por correlación buena todos aquellos valores que sean mayores a 0.7 o

menores a -0.7. Como es normal, la diagonal de la matriz es 1 debido a que se calcula el coeficiente de *Spearman* sobre el mismo bloque, es decir, tienen una correlación perfecta (son idénticos).

$$\begin{bmatrix} 1 & -0{,}33 & 0{,}03 & 0{,}43 & -0{,}12 & -0{,}01 & 0{,}05 & -0{,}02 & -0{,}11 & -0{,}08 \\ -0{,}33 & 1 & -0{,}34 & -0{,}22 & 0{,}01 & 0{,}02 & 0{,}32 & 0{,}36 & 0{,}34 & 0{,}27 \\ 0{,}03 & -0{,}34 & 1 & 0{,}12 & 0{,}20 & -0{,}51 & -0{,}70 & -0{,}45 & -0{,}31 & -0{,}02 \\ 0{,}43 & -0{,}22 & 0{,}12 & 1 & 0{,}10 & 0{,}03 & -0{,}13 & -0{,}18 & -0{,}41 & -0{,}42 \\ -0{,}12 & 0{,}01 & 0{,}20 & 0{,}10 & 1 & -0{,}22 & -0{,}37 & -0{,}52 & -0{,}49 & -0{,}22 \\ -0{,}01 & 0{,}02 & -0{,}51 & 0{,}03 & -0{,}22 & 1 & 0{,}46 & 0{,}18 & -0{,}01 & -0{,}32 \\ 0{,}05 & 0{,}32 & -0{,}70 & -0{,}13 & -0{,}37 & 0{,}46 & 1 & 0{,}53 & 0{,}41 & 0{,}12 \\ -0{,}02 & 0{,}36 & -0{,}45 & -0{,}18 & -0{,}52 & 0{,}18 & 0{,}53 & 1 & 0{,}57 & 0{,}30 \\ -0{,}11 & 0{,}34 & -0{,}31 & -0{,}41 & -0{,}49 & -0{,}01 & 0{,}41 & 0{,}57 & 1 & 0{,}45 \\ -0{,}08 & 0{,}27 & -0{,}02 & -0{,}42 & -0{,}22 & -0{,}32 & 0{,}12 & 0{,}30 & 0{,}45 & 1 \end{bmatrix}$$

Figura 5.9: Matriz simétrica 10×10 de correlación de *Spearman*

Como observamos en el resultado matricial, existen dos bloques que mantienen un comportamiento similar. En este caso es el bloque 2 y el 6 con un valor absoluto de 0,70. Si observamos otra vez la figura 5.7 se puede apreciar que éstos bloques efectivamente mantienen un comportamiento similar.

5.3.2. Comparación de volatilidad entre bloques

En nuestro caso utilizaremos la volatilidad para intentar establecer una relación entre los modelos y la efectividad que tienen dependiendo de ésta medida.

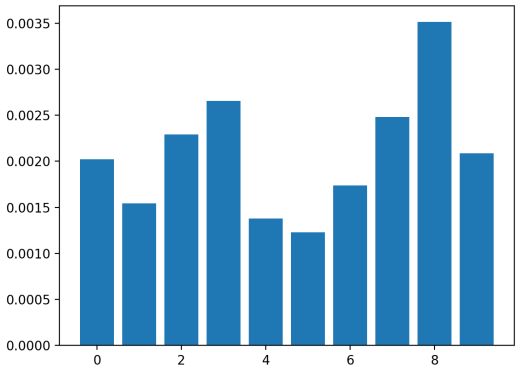

Figura 5.10: Volatilidad media de las características de los bloques (eje x)

5.3.3. Comparación de volatilidad entre conjuntos dentro de los bloques

Sacamos la diferencia de volatilidad de los conjuntos train y test de cada bloque.

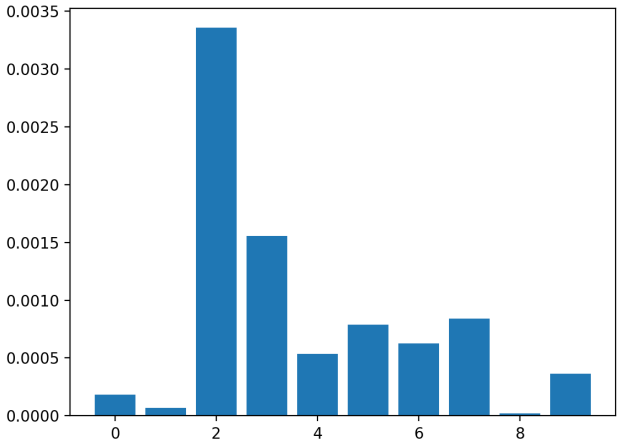

Figura 5.11: Diferencia de volatilidad entre el conjunto de train y test de cada uno de los bloques.

Capítulo 6

Metodología

Como vimos en el apartado anterior de análisis, utilizaremos el *Blocked Cross-Validation* para que los resultados obtenidos sean independientes de los datos con los que experimentamos. En este capítulo veremos los métodos que aplicamos para intentar solucionar los problemas planteados en este trabajo.

6.1. Conjunto de train, test y validación

Cada bloque del Cross-Validation se divide a su vez en tres partes: el conjunto de entrenamiento, el de validación y el de test.

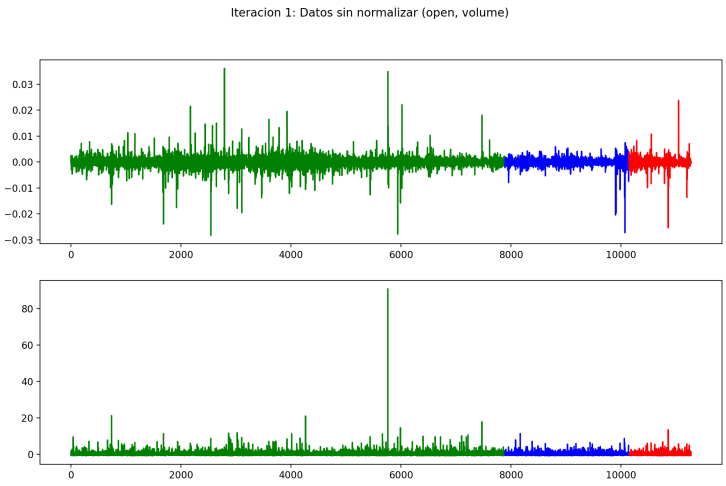

Figura 6.1: Conjunto de entrenamiento *(verde)*, validación *(azul)* y test *(rojo)*.

La proporción en la que separamos el bloque es importante ya que cuanto menos datos de entrenamiento tengas, menor será la posibilidad de que el comportamiento del

mercado en ese instante sea el mismo que el que ocurre en el conjunto de validación o test.

En este trabajo se ha decido escoger los valores estándar: entrenamiento 70 %, validación 20 % y test 10 % (cuadro 6.1).

6.2. Clasificación de modelos por tipo de predicción

En la experimentación, vamos a comparar el rendimiento que ofrecen los modelos en dos formas de predecir. Dependiendo de ésta forma, los modelos los clasificamos en: **modelos de múltiple paso** o **de paso a paso**.

Si el modelo es de paso a paso y tenemos que hacer una predicción de N velas, podemos hacerlo en N iteraciones generando una vela en cada una de éstas tal y como se muestra en la figura siguiente (figura 6.2).

Figura 6.2: Ejemplo iteraciones modelo paso a paso.

Si el modelo es de múltiple paso la predicción se hace en una sola iteración tal y como muestra la siguiente figura (figura 6.3).

Figura 6.3: Ejemplo iteraciones modelo de múltiple paso.

6.3. Entrenamiento y evaluación de los modelos

En este apartado abordaremos la metodología empleada para poder comparar modelos de múltiple paso con modelos paso a paso.

Si el modelo es paso a paso, el número de salida de velas de la red será 1 y tendrá que hacer n pasos que coincidirán con el número de velas a predecir (figura 6.4).

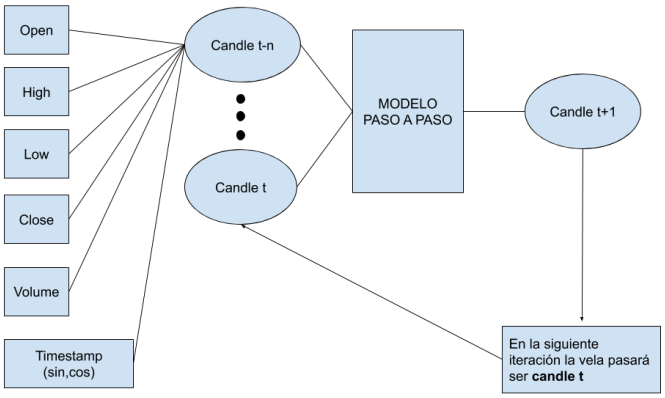

Figura 6.4: Diagrama de funcionamiento del modelo paso a paso.

Si es un modelo de múltiple paso, el número de salida de velas de la red será el mismo que el número de velas a predecir por la red tal y como muestra la figura 6.5.

Una vez tenemos las predicciones realizadas en ambos modelos, podremos comparar esa predicción usando la función de pérdida *Mean Absolute Error* (**MAE**).

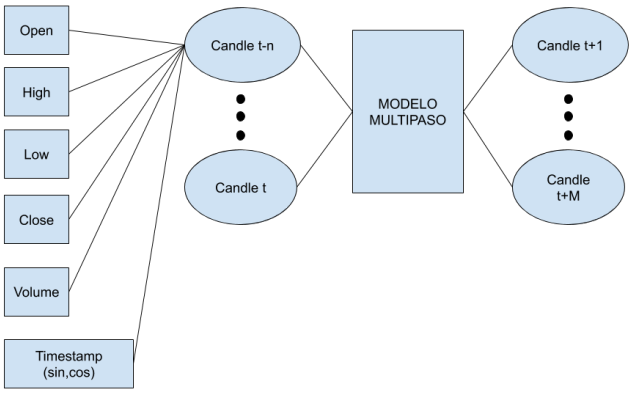

Figura 6.5: Diagrama de funcionamiento del modelo de múltiples pasos.

6.4. Backtesting

Desde el principio de este documento se ha explicado al lector que este estudio está enfocado a elaborar una estrategia para operar el mercado de valores y no, únicamente, a una aproximación científica. Si nos quedásemos simplemente con el valor de error que obtenemos de los modelos experimentados, no se podría concluir que los modelos con un valor determinado de error sean eficientes sacando rendimiento del mercado bursátil.

Estas conclusiones serán incorrectas porque en el mercado de valores no solo dependes de acertar el movimiento del mercado sino que hay otra parte muy importante que es la **gestión de riesgo**. Como vimos en el capítulo de introducción bursátil[2], una operación tiene que tener su *stoploss* para gestionar grandes movimientos en bolsa que puedan jugar en tu contra o, simplemente, que no hayas acertado el movimiento del mercado y tengas que salir. Esa gestión también dependerá de otros factores como la cantidad de entrada al mercado, apalancamiento, comisiones, etc.

Tan complejo es este problema que existe un campo de investigación dedicado a la gestión de operaciones. Dentro de éste, se estudian algoritmos de *Machine Learning* que ayuden a minimizar el riesgo de éstas.

Existen multitud de librerías que nos pueden ayudar en la tarea de testear algoritmos dado un historial de precios y volumen. En nuestro caso, hemos implementado el *backtesting* con una estrategia muy simple que nos permita evaluar los modelos estudiados.

Los datos a predecir serán los del conjunto test de cada iteración. Según el resultado del sumatorio de las predicciones de velas de la red, es decir, cuanto se desplaza el precio, entraremos en el mercado a corto (vendiendo) o a largo (comprando). Todo esto con su gestión de comisiones y *stoploss*. El portafolio inicial será de 1000 unidades y la entrada en el mercado será el 10 % del capital del portafolio en cada momento. Como los datos están sacados del *exchange* de Binance, las comisiones serán las mismas y se aplican a la entrada y a la salida del mercado, siendo 0.04 % éstas.

El *backtesting* nos devolverá la variación de nuestro portafolio en porcentaje y el *drawdown* máximo al acabar. El *stoploss* será del 3 % para evitar pérdidas grandes.

Una vez acabado el *backtesting*, podremos comparar los modelos de dos formas diferentes: error generado medio y rendimiento del sistema.

Capítulo 7

Experimentación

Primero vamos a definir los modelos que se han utilizado para la experimentación. Partiremos desde los modelos más sencillos (modelos *baseline*, lineales y no lineales simples) hasta los modelos más complejos, valorando diferentes parámetros, para compararlos el entre ellos.

En todos los modelos con los que se experimenta se usa la función *loss* de **error cuadrático medio** mientras que el optimizador es ***Adam***. Los resultados de error de los cuadros expuestos están en **error absoluto medio**.

7.1. Baseline

El modelo Baseline se trata de definir un modelo base el cual podamos comparar con los demás modelos para ver si realmente obtienen mejora respecto a no predecir nada.

Para ello, el modelo lo único que hace es predecir que las velas siguientes sean igual a la vela anterior, es decir, no varía el precio respecto al tiempo.

7.1.1. BaselinePerc

En este caso, el modelo hace las predicciones de retorno de precio con valor cero. Esto es así porque el valor del precio no debe de cambiar, es decir, el retorno en cada vela debe ser cero.

```
1 class BaselinePerc(tf.keras.Model):
2     def __init__(self, num_output):
```

```
        super().__init__()
        self.num_output = num_output

    def call(self, inputs):
        return tf.tile(tf.zeros_like(inputs[:, -1:, :]),
                       [1, self.num_output, 1])
```

Listing 7.1: Implementación del modelo BaselinPerc en Python con Keras

7.1.2. BaselineAbs

Otro modelo de tipo Baseline es hacer que el retorno del precio se mantenga con el valor de la última entrada a la red.

```
class BaselineAbs(tf.keras.Model):
    def __init__(self, num_output):
        super().__init__()
        self.num_output = num_output

    def call(self, inputs):
        return tf.tile(inputs[:, -1:, :], [1, self.num_output, 1])
```

Listing 7.2: Implementación del modelo BaselinAbs en Python con Keras

7.2. Modelos lineales

7.2.1. Regresión lineal

```
def LinearModel(features=1, steps=1):
    return tf.keras.Sequential([
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(steps*features, activation='linear'),
        tf.keras.layers.Reshape([steps, features])
    ])
```

Listing 7.3: Implementación del modelo regresión lineal en Python con Keras

7.3. Modelos no lineales

Como modelos no lineales tendremos a las redes neuronales de los tipos que vimos en el Capitulo 4. Para ello empezaremos con los modelos de redes neuronales más básicos e iremos juntando capas de distintos tipos de neuronas.

7.3.1. Modelos simples

7.3.1.1. Modelo MLP

```
def MLPModel(features=1, steps=1):
    return tf.keras.Sequential([
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation='tanh'),
        tf.keras.layers.Dense(256, activation='tanh'),
        tf.keras.layers.Dense(128, activation='tanh'),
        tf.keras.layers.Dense(steps*features, activation='tanh'),
        tf.keras.layers.Reshape([steps, features])
    ])
```

Listing 7.4: Implementación del modelo MLP en Python con Keras

7.3.1.2. Modelo LSTM

```
def LSTMModel(features=1, steps=1):
    return tf.keras.Sequential([
        tf.keras.layers.LSTM(32,
                             return_sequences=False,
                             activation='tanh'),
        tf.keras.layers.Dense(steps*features, activation='tanh'),
        tf.keras.layers.Reshape([steps, features])
    ])
```

Listing 7.5: Implementación del modelo LSTM en Python con Keras

7.3.1.3. Modelo GRU

```
def GRUModel(features=1, steps=1):
    return tf.keras.Sequential([
        tf.keras.layers.GRU(32, return_sequences=False, activation='
    tanh'),
```

```
        tf.keras.layers.Dense(steps*features, activation='tanh'),
        tf.keras.layers.Reshape([steps, features])
    ])
```

Listing 7.6: Implementación del modelo GRU en Python con Keras

7.3.2. Modelos complejos

7.3.2.1. Modelo CNN-MLP

```
def Model_CNN_MLP(features=1, steps=1):
    return tf.keras.Sequential([
        # layer cnn
        tf.keras.layers.Conv1D(filters=32,
                    kernel_size=(3,),
                    activation='tanh'),

        tf.keras.layers.Flatten(),

        # layer Dense
        tf.keras.layers.Dense(512, activation='tanh'),
        tf.keras.layers.Dense(256, activation='tanh'),
        tf.keras.layers.Dense(128, activation='tanh'),

        tf.keras.layers.Dense(steps*features, activation='tanh'),
        tf.keras.layers.Reshape([steps, features])
    ])
```

Listing 7.7: Implementación del modelo CNN-MLP en Python con Keras

7.3.2.2. Modelo RNN-MLP

```
def Model_RNN_MLP(features=1, steps=1):
    return tf.keras.Sequential([

        # layer RNN
        tf.keras.layers.LSTM(128, return_sequences=True, activation='
    tanh'),
        tf.keras.layers.LSTM(64, return_sequences=True, activation='
    tanh'),
        tf.keras.layers.LSTM(32, return_sequences=False, activation='
    tanh'),
```

```
8
9           # layer Dense
10          tf.keras.layers.Dense(512, activation='tanh'),
11          tf.keras.layers.Dense(256, activation='tanh'),
12          tf.keras.layers.Dense(128, activation='tanh'),
13
14          #layer output
15          tf.keras.layers.Dense(steps*features, activation='linear'),
16
17          tf.keras.layers.Reshape([steps, features])
18      ])
```

Listing 7.8: Implementación del modelo LSTM-MLP en Python con Keras

7.3.2.3. Modelo CNN-RNN

```
1  def Model_CNN_RNN(features=1, steps=1):
2      return tf.keras.Sequential([
3          # layer cnn
4          tf.keras.layers.Conv1D(filters=32,
5                                 kernel_size=(3,),
6                                 activation='tanh'),
7
8          # layer RNN
9          tf.keras.layers.LSTM(128, return_sequences=True, activation='
       tanh'),
10         tf.keras.layers.LSTM(64, return_sequences=True, activation='
       tanh'),
11         tf.keras.layers.LSTM(32, return_sequences=False, activation='
       tanh'),
12
13         # layer Dense
14         tf.keras.layers.Dense(512, activation='tanh'),
15
16         tf.keras.layers.Dense(steps*features, activation='tanh'),
17         tf.keras.layers.Reshape([steps, features])
18     ])
```

Listing 7.9: Implementación del modelo CNN-LSTM-MLP en Python con Keras

7.3.2.4. Modelo CNN-RNN-MLP

```
def Model_CNN_RNN_MLP(features=1, steps=1):
    return tf.keras.Sequential([
        # layer cnn
        tf.keras.layers.Conv1D(filters=32,
                    kernel_size=(3,),
                    activation='tanh'),

        # layer RNN
        tf.keras.layers.LSTM(128, return_sequences=True, activation='
    tanh'),
        tf.keras.layers.LSTM(64, return_sequences=True, activation='
    tanh'),
        tf.keras.layers.LSTM(32, return_sequences=False, activation='
    tanh'),

        # layer Dense
        tf.keras.layers.Dense(512, activation='tanh'),
        tf.keras.layers.Dense(256, activation='tanh'),
        tf.keras.layers.Dense(128, activation='tanh'),

        #layer output
        tf.keras.layers.Dense(steps*features, activation='tanh'),
        tf.keras.layers.Reshape([steps, features])
    ])
```

Listing 7.10: Implementación del modelo CNN-LSTM-MLP en Python con Keras

7.3.2.5. Modelo TCN1

Este modelo solo tiene una capa TCN.

```
def TCN1Model(inputs, features, steps):
    i = Input(batch_shape=inputs)
    o = TCN(return_sequences=False)(i)
    o = Dense(steps*features, activation='tanh')(o)
    o = Reshape([steps, features])(o)
    return Model(inputs=[i], outputs=[o])
```

Listing 7.11: Implementación del modelo TCN en Python con Keras

7.3.2.6. Modelo múltiples series unificadas

Hemos elaborado un modelo que primero hace una predicción de cada una de las características de la vela (*open*, *high*, *low*, *close* y *volume*) por separado. Posteriormente unifica esas predicciones y modifica aquellos valores con la dependencia que les toca (figura 7.1).

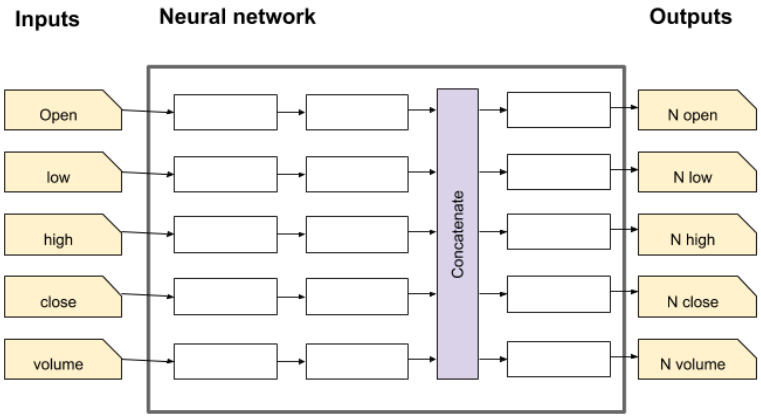

Figura 7.1: Diagrama del modelo.

```
def GRU_Layer(inputs, features, steps, name):
    i = Input(shape=inputs, name=name)
    o = tf.keras.layers.GRU(steps*features,
                            return_sequences=False,
                            activation='tanh')(i)
    o = Reshape([steps, features])(o)

    return Model(inputs=i, outputs=o)


def CustomModel(inputs, features, steps):

    open_serie = GRU_Layer(inputs,
                           features=1,
                           steps=steps,
                           name='open')

    high_serie = GRU_Layer(inputs,
                           features=1,
                           steps=steps,
```

```
                        name='high')

    low_serie = GRU_Layer(inputs,
                        features=1,
                        steps=steps,
                        name='low')

    close_serie = GRU_Layer(inputs,
                        features=1,
                        steps=steps,
                        name='close')

    volume_serie = GRU_Layer(inputs,
                        features=1,
                        steps=steps,
                        name='volume')

    # combine the output
    combined = concatenate([open_serie.output,
                        high_serie.output,
                        low_serie.output,
                        close_serie.output,
                        volume_serie.output])

    z = GRU_Layer((steps, features),
                features=features,
                steps=steps,
                name='default') (combined)
    o = Reshape([steps, features])(z)

    return Model(inputs=[open_serie.input,
                        high_serie.input,
                        low_serie.input,
                        close_serie.input,
                        volume_serie.input], outputs=o)
```

Listing 7.12: Implementación del modelo complejo en Keras

7.4. Resultados modelos simples sin normalización de datos

Estos resultados generales se presentan agrupados por modelo haciendo una media del error, portafolio y drawdown de todas las iteraciones. Los modelos tienen como input 50 velas y como output 5.

modelo	MAE	portafolio (%)	drawdown (%)
baselineabs	0.39	2.30	0.24
lstm_multistep	0.38	1.77	0.37
gru_multistep	0.38	1.38	0.59
linear_multistep	0.38	0.67	0.85
gru_step	0.38	0.00	1.13
baselineperc	0.39	0	0
linear_step	0.38	−0.06	0.91
lstm_step	0.38	−0.65	1.46
mlp_step	0.38	−1.07	1.76
mlp_multistep	0.38	−1.64	1.76

Cuadro 7.1: Resultados modelos simples sin datos normalizados

Los modelos lineales son los que mejor rendimiento sacan del mercado en este experimento. También vemos como no hay relaciones en pequeñas variaciones del error y el portafolio final.

7.5. Resultados modelos simples con normalización de datos

En la figura 6.1 veíamos los valores de retorno pasados a los modelos. En este caso lo que haremos será la misma prueba que en el anterior apartado pero normalizando los datos. Para ello usaremos diferentes algoritmos de normalización. El número de velas de entrada y salida de los modelos son las mismas que en el apartado anterior.

7.5.1. *QuantileTransformer*

Utilizaremos el escalador *QuantilTransformer* para normalizar los datos. Para hacerlo de la forma adecuada, transformaremos los datos de test a partir de los datos de train. En la siguiente figura podemos ver las diferencias entre los datos sin escalar y con el escalado mencionado.

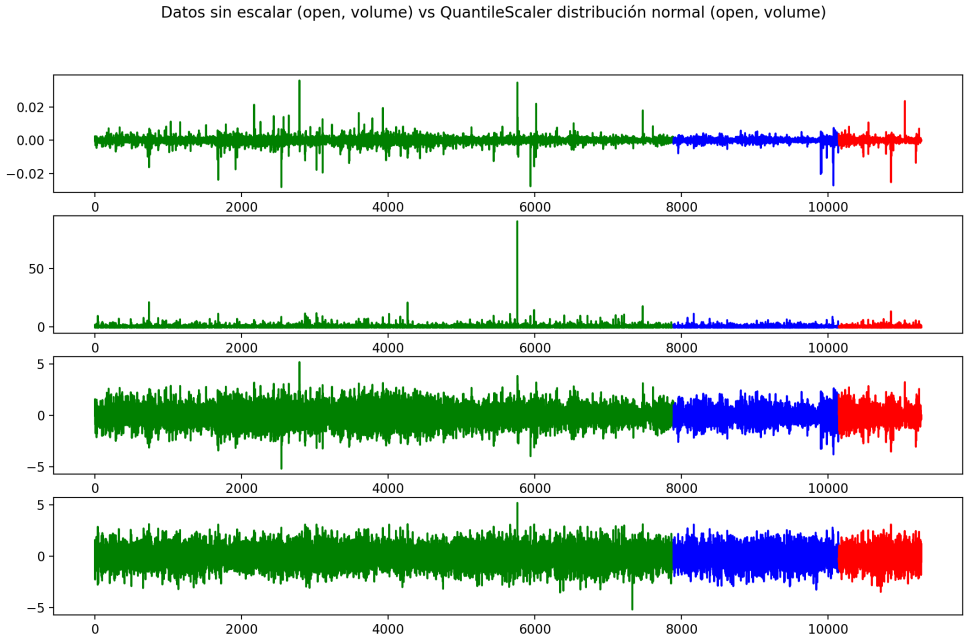

Figura 7.2: Conjunto de entrenamiento *(verde)*, validación *(azul)* y test *(rojo)*.

modelo	MAE	portafolio (%)	drawdown (%)
gru_multistep	0.30	3.61	0.15
lstm_multistep	0.30	3.52	0.16
mlp_step	0.30	3.50	0.16
lstm_step	0.30	3.48	0.16
linear_multistep	0.30	3.45	0.15
mlp_multistep	0.30	3.45	0.16
gru_step	0.30	3.44	0.16
linear_step	0.30	3.43	0.18
baselineabs	0.30	2.30	0.24
baselineperc	0.31	0	0

Cuadro 7.2: Resultados modelos simples con datos normalizados con QuantileTransformer

Comparando éstos datos con los de la tabla anterior, existe una relación entre menor error en los modelos y mayor rendimiento de éste.

7.5.2. *PowerTransform*

Ahora compararemos el anterior escalador con *PowerTransform*.

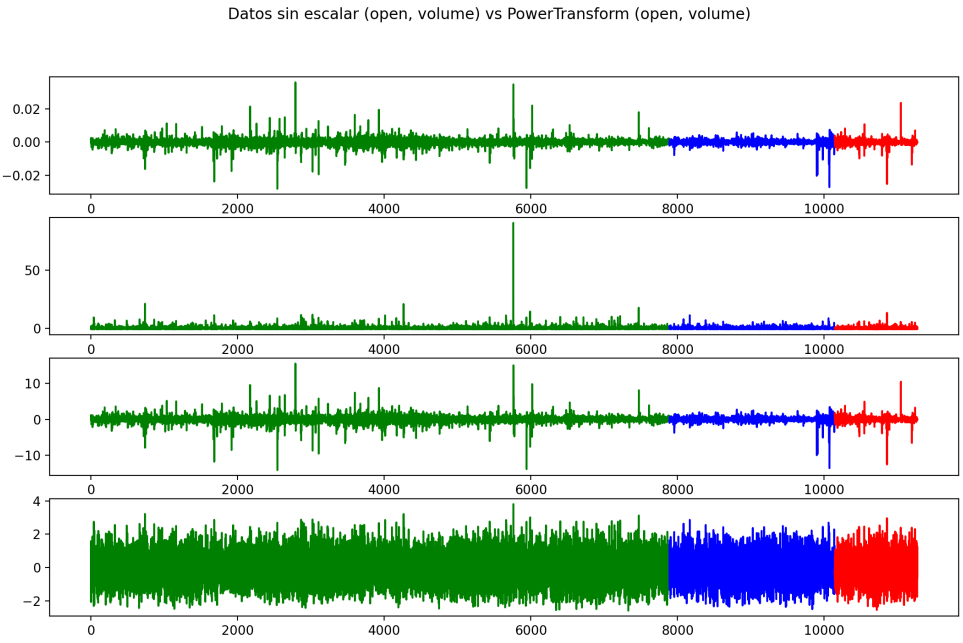

Figura 7.3: Conjunto de entrenamiento *(verde)*, validación *(azul)* y test *(rojo)*.

modelo	MAE	portafolio (%)	drawdown (%)
lstm_multistep	0.99	3.26	0.13
gru_multistep	0.99	3.22	0.14
linear_multistep	0.99	3.09	0.22
mlp_multistep	0.99	3.04	0.21
gru_step	0.99	3.02	0.21
lstm_step	0.99	2.59	0.16
linear_step	0.99	2.59	0.21
mlp_step	0.99	2.51	0.27
baselineabs	0.99	2.30	0.24
baselineperc	1.00	0	0

Cuadro 7.3: Resultados modelos simples con datos normalizados con PowerTransform

Comparando éstos datos con los de la tabla anterior, existe una relación entre menor error en los modelos y mayor rendimiento de éste. Además, el escalador *QuantileTransformer* da mejores resultados que éste ya que los resultados en el anterior apartado son más uniformes.

7.6. Resultados modelos complejos con normalización de datos

Ahora estudiaremos el rendimiento de los modelos complejos descritos anteriormente. En este caso se ha utilizado para normalizar los datos *QuantileTransformer* ya que ha dado mejor rendimiento en el apartado anterior.

modelo	MAE	portafolio (%)	drawdown (%)
rnnmlp_multistep	0.26	3.45	0.15
cnnrnnmlp_multistep	0.26	3.42	0.14
cnnmlp_multistep	0.26	3.40	0.17
cnnrnn_multistep	0.26	3.38	0.13
tcn1_multistep	0.26	−0.49	0.27

Cuadro 7.4: Resultados modelos complejos con datos normalizados

Los modelos complejos no consiguen un mejor rendimiento exceptuando el modelo *rnnmlp_multistep*.

7.7. Resultados modelos con cambios en parámetros de entrada y salida

Vamos a comparar diferentes parámetros de entrada (50, 100 y 200 velas) y salida (5, 10 y 15 velas) con los modelos que mejor rendimiento nos ha dado anteriormente.

input	output	MAE	portfolio (%)	drawdown (%)
50	15	0.29	2.89	0.16
50	5	0.30	2.82	0.18
50	10	0.26	2.81	0.17
100	15	0.28	2.16	0.23
100	10	0.26	2.15	0.25
100	5	0.29	2.00	0.18
200	10	0.26	1.35	0.38
200	5	0.29	1.29	0.40
200	15	0.27	1.03	0.24

Cuadro 7.5: Resultados según parámetros de entrada y salida

Podemos observar en el cuadro 7.5 que haciendo una media de los resultados agru-

pados por input y output los modelos que mejor resultado de rendimiento (portafolio) nos da son los de input 50 y output 15. Además cuanto mayor son los parámetros de entrada a la red mayor drawdown conseguimos.

7.8. Experimentación CustomModel

Aplicando las conclusiones de los apartados anteriores, vamos a utilizar el *CustomModel* para ver si éste podría obtener mejor rendimiento del mercado.

El propósito es usar las primeras capas del modelo para que haga una predicción de cada una de las series temporales (*open, high, low, close y volume*) sin que tenga en cuenta las relaciones entre ellas. Posteriormente se concatenarán estas predicciones a otra capa para que modifique las salidas teniendo en cuenta la relación entre éstas.

En el cuadro 7.6, el modelo que más rendimiento sacaba era el *gru_multistep*. *CustomModel* estará compuesto integramente por estas capas (subsección 7.3.2.6).

modelo	MAE	portafolio (%)	drawdown (%)
custom_model_multistep	0.24	−1.11	1.38

Cuadro 7.6: Rendimiento medio del CustomModel.

En este caso el modelo *custom* no mejora el rendimiento ofrecido por otros modelos, sin embargo, el tiempo de entrenamiento si que aumenta considerablemente.

7.9. Comparación de modelos con datos volátiles

En el estudio de volatilidad (subsección 5.3.2) y de diferencia de volatilidad (subsección 5.3.3) entre el *train* y *test* de las iteraciones, observamos como la **iteración 8** (figura 5.10 y 5.11) es la iteración más volátil a la par que es la que menos diferencia de volatilidad tiene entre su conjunto *train* y *test*. Los resultados de los modelos en esa iteración se muestran en el siguiente cuadro (cuadro 7.8).

Podemos comparar estos resultados con la **iteración 2** (figura 5.11) donde la variación de volatilidad entre el conjunto *test* y el *train* es la máxima de todas las iteraciones.

model	MAE	portfolio (%)	drawdown (%)
baselineperc	0.35	0	0
baselineabs	0.34	5.07	0.55
linear_multistep	0.29	7.56	0.49
mlp_multistep	0.29	7.57	0.49
lstm_multistep	0.29	7.43	0.60
gru_multistep	0.29	7.55	0.49
cnnmlp_multistep	0.29	7.17	0.30
cnnrnn_multistep	0.29	7.20	0.31
rnnmlp_multistep	0.29	7.32	0.49
cnnrnnmlp_multistep	0.29	7.26	0.31
tcn1_multistep	0.29	−2.74	2.44
custom_model_multistep	0.31	−2.65	4.02

Cuadro 7.7: Resultados modelos en iteración 8 aplicado *QuantileTransform* a los datos

model	MAE	portfolio (%)	drawdown (%)
baselineperc	0.31	0	0
baselineabs	0.30	1.59	0.22
linear_multistep	0.29	3.08	0
mlp_multistep	0.29	3.14	0
gru_multistep	0.30	3.17	0
lstm_multistep	0.29	3.12	0
custom_model_multistep	0.22	−0.60	1.00
cnnrnn_multistep	0.29	2.63	0.02
cnnmlp_multistep	0.25	2.96	0.02
rnnmlp_multistep	0.29	3.12	0
cnnrnnmlp_multistep	0.29	2.92	0.02
tcn1_multistep	0.29	−0.31	0

Cuadro 7.8: Resultados modelos en iteración 2 aplicado *QuantileTransform* a los datos

Capítulo 8

Conclusiones y trabajos futuros

En este trabajo se han abordado una serie de objetivos específicos que se realizaron durante el desarrollo de este proyecto y que han culminado en los siguientes conclusiones:

1. Los modelos neuronales tienen una mejora considerable cuando se normalizan los datos de retorno de las velas. Se ha comparado *QuantileTransforme* con *PowerTransform* obteniendo mejores resultados con el primero.

2. Los modelos paso a paso y múltiple paso sacan un rendimiento similar. Como el coste temporal en los de múltiple paso era menor, se ha optado por ellos en el resto de pruebas.

3. La arquitectura TCN no ha dado buen rendimiento comparado con las otras arquitecturas.

4. Hay que hacer pruebas con diferentes valores de entrada y salida a la red para obtener un valor óptimo. Muchos parámetros de entrada y escasos valores de salida pueden generar un incremento de *drawdown* y viceversa.

5. Los modelos neurales dan mejor rendimiento cuando la diferencia de volatilidad entre el conjunto *train* y *test* es mínima. Una línea de trabajo futuro podría ser analizar las velas de cotización actual y buscar tramos con alta correlación para entrenar los modelos con esos tramos.

En este trabajo hemos abordado una de las múltiples formas de intentar predecir los mercados financieros con sistemas neurales. Por lo que se refiere a líneas futuras

a corto plazo, podemos aplicar estos mismos experimentos a otros mercados financieros para ver si los modelos consiguen un resultado similar. Además, podemos aplicar estos algoritmos a librerías de *backtesting* robustas como es **Backtrader**, utilizando diferentes analizadores de mercado que contiene este *framework* para analizar con más profundidad éstos modelos.

En lo que se refiere a líneas futuras a largo plazo, estos modelos neuronales podrían ir de la mano de estrategias tradicionales para complementar la señal de entrada o salida al mercado. También se podrían aplicar modelos de gestión de riesgo para minimizar pérdidas y maximizar ganancias.

Para finalizar, en lo que respecta a abordar éste problema con modelos neuronales, un campo interesante de estudio sería aplicar algoritmos de aprendizaje reforzado a éstas arquitecturas para que el modelo se vaya adaptando al mercado dependiendo de la recompensa (cantidad de retorno) que genere.

References

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoderfor statistical machine translation.

Doshi, A., Rafati, S., Issa, A., Rakshit, S., and Sachdeva, P. (2020). Deep stock predictions.

Elend, L., Tideman, S. A., Lopatta, K., and Kramer, O. (2020). Earnings prediction with deep learning.

Healton, J. B., Polson, N. G., and Witte, J. H. (2018). Deep learning in finance.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory.

Liu, J., Chao, F., Lin, Y.-C., and Lin, C.-M. (2019). Stock prices prediction using deep learning models.